CHALMERS
UNIVERSITY OF TECHNOLOGY

Design and Formal Verification of a Safe Stop Supervisor for an Automated Vehicle

2019 International Conference on Robotics and Automation (ICRA)
Palais des congres de Montreal, Montreal, Canada, May 20-24, 2019

Design and Formal Verification of a Safe Stop Supervisor for an Automated Vehicle*

Jonas Krook[1,3] Lars Svensson[2] Yuchao Li[2] Lei Feng[2] Martin Fabian[3]

Abstract— **Autonomous vehicles apply pertinent planning and control algorithms under different driving conditions. The mode switch between these algorithms should also be autonomous. On top of the nominal planners, a safe fallback routine is needed to stop the vehicle at a safe position if nominal operational conditions are violated, such as for a system failure. This paper describes the design and formal verification of a supervisor to manage all requirements for mode switching between nominal planners, and additional requirements for switching to a safe stop trajectory planner that acts as the fallback routine. The supervisor is designed via a model-based approach and its abstraction is formally verified by model checking. The supervisor is implemented and integrated with the Research Concept Vehicle, an experimental research and demonstration vehicle developed at the KTH Royal Institute of Technology. Simulations and experiments show that the vehicle is able to autonomously drive in a safe manner between two parking lots and can successfully come to a safe stop upon GPS sensor failure.**

I. Introduction

The past three decades have witnessed a joint effort between academia and industry to realize driverless vehicles. The SAE design guideline J3016 [1] classifies automated road vehicles into six different categories ranging from "No Driving Automation" to "Full Driving Automation". At present, the automotive industry is aiming toward deploying level 4 functionality ("High Driving Automation") [2], [3] i.e., full automation on selected parts of the traffic network. Since level 4 automation allows operation of a vehicle without a human operator being present, this is where automation can really impact society in terms of how people and goods are transported. For a level 4 automated vehicle, the control system is required to fulfill the driving task during nominal operational conditions, as well as automatically reach a so called *minimal risk condition* when nominal operational conditions are violated. For example, if an event occurs that prevents completion of the current mission, the vehicle should automatically stop, preferably outside of active traffic lanes. Since there is no human operator, the safety and correctness of the vehicles' on-board systems are of utmost importance. State of the art in the field [2], [3] suggests that redundancy and fallback routines should be applied such that a safe stop, i.e., a maneuver that realizes the minimal risk condition, can be executed at all times, and at high level of integrity.

To reach such a high level of integrity, there is a need to alleviate the risk of faults occurring in the implemented software. Moreover, there is a need to provide convincing evidence that the probability of fault occurrences is low. The ISO 26262 standard [4], dedicated to automotive safety, points to several methods that are available to this end. For instance, research has shown that model-driven software engineering can improve the product reliability [5]. Since the safe stop routine must be highly dependable, the correctness of the supervisor must be supported by convincing and rigorous evidence. Such evidence is not attained by model-driven software development alone. The research fields of supervisory control theory [6] and model checking [7], have developed tools for proving properties of such a supervisor [8]. Since these tools either prove the correctness of software or identify counterexamples, they provide very convincing evidence.

Model-based design, model checking, and safe stop routines have been considered for automotive applications in earlier research. For instance [9] successfully applies model-based design and formal methods in order to synthesize and prove the correctness of the switching logic between manual drive, cruise-control, and adaptive cruise-control. The usefulness of formal verification of manually developed code for vehicles has also been shown [10]–[12]. Within trajectory planning, [13] and [14] have made contributions in the area of safe stop routines. In [15], an emergency maneuver routine is introduced that is similar to the approach in this paper, but the maneuver is not required to stop the vehicle, and in-lane stopping is not actively avoided. The aforementioned research does not, however, consider the combination of a formally verified supervisor and a safe stop routine.

This paper extends previous research by introducing a safe stop routine and evaluating a systematic approach to guarantee correctness of the supervisor by formal verification. Model-based techniques are used to design and test the supervisor that coordinates the operation of nominal planning algorithms and the safe stop routine. The supervisor and a model of the test vehicle system are implemented and formally verified. Compared to [9], this paper manually designs the supervisory controller and verifies it by model checking, whereas [9] automatically synthesizes the supervisor from the specifications modeled by finite state automata. The advantage of our method is that more expressive specifications formalized in linear temporal logic are considered.

Our test vehicle is required to drive autonomously between

*This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation; and Integrated Transport Research Lab (ITRL).

[1]Zenuity, Gothenburg, Sweden jonas.krook@zenuity.com
[2]KTH, Stockholm, Sweden
{larsvens,yuchao,lfeng}@kth.se
[3]Chalmers, Göteborg, Sweden
{krookj,fabian}@chalmers.se

978-1-5386-6026-3/19/$31.00

two parking lots via a road network. It is demonstrated through simulations and real experiments that the safe stop routine can safely guide the vehicle to a stop by the side of the road during the mission. The supervisor correctly enables the mode switching actions so that the correct control functions are applied. In particular, the safe stop command is issued if and only if it is needed.

The outline of this paper is as follows. Section II presents an overview of the system architecture for which the supervisor is designed and integrated. Also, relevant trajectory planning and control functions are briefly presented. Then, Section III is dedicated to the design and implementation of the proposed supervisor. Section IV presents, interprets and discusses the results of the case study. Finally, conclusions and future work are presented in Section V.

II. System Architecture

The KTH Research Concept Vehicle (RCV) is a custom-built, fully electric and drive-by-wire concept vehicle, hosted by the Integrated Transport Research Lab at KTH, for validating and demonstrating research results [16]–[18]. The vehicle is equipped with cameras, lidars and GPS for perception and localization. In addition to the physical vehicle, a Gazebo model of the RCV, extended from [19], is part of the development environment.

The software system of the RCV is split into different components, which are specialized and responsible for distinct sub-functionalities. The components are implemented as separate software nodes in the Robot Operating System (ROS) [20] and communicate with each other. Figure 1 shows the system architecture. Boxes indicate the software nodes, and arrows indicate signal flow. The system is built in a hierarchical way, where the Supervisor is the top-most and central software node.

The task of the Supervisor component is to issue commands to the components of the system such that the vehicle fulfills its mission, in this case to safely bring the RCV from a starting position in a parking lot to a goal position in another parking lot, via the road network. When the goal position is safely reached and the RCV is stationary, then the mission successfully terminates and a new mission may be issued. Extensions of this (e.g. starting in the road network, or receiving new goals while driving) might be desired in the future, but are not considered in this paper.

The *Localization* component provides the Supervisor with current position information. It also communicates the goal position, and detects and reports GPS sensor failures.

The *Structured Area Path Planner* (SPP) generates a *path* through the road network by searching for the shortest route through an efficient map representation [21]. The center of the lane in the selected route is extracted and transformed to the path format, taking nonholonomic constraints into account. A path is a spatial reference line for the RCV to follow. Two transition points are associated with the path, Tp1 and Tp2, which are the positions on the boundary between the road and the source and goal parking lot, respectively. It is assumed that when a request to plan is

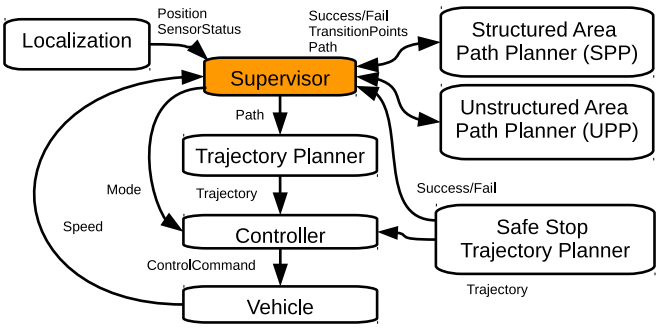


Fig. 1: ROS component architecture. Boxes indicate separate software nodes, while arrows indicate signal flow.

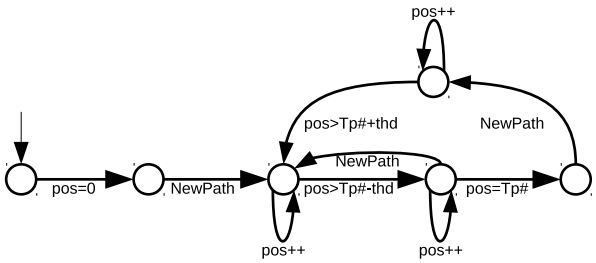


Fig. 2: FSM for the Trajectory Planner component. `pos` is the current position along the path. Tp# is either Tp1, Tp2, or Goal position, initiated to Tp1 and updates to the next as thresholds are passed. `thd` is an arbitrary threshold that prevents erroneous state switching caused by sensor noise.

received from the Supervisor, SPP responds with success or fail depending on whether a path could be found or not.

The *Unstructured Area Path Planner* (UPP) provides paths in unstructured environments, e.g., parking lots. Other than obstacle avoidance and the RCV's nonholonomic constraints, unstructured areas do not have any imposed restrictions on how the vehicle may move within them. The implementation [22] is based on a Hybrid A* algorithm [23]. It is assumed that when UPP receives from the Supervisor a request to plan, the outcome of the planning results in success or failure, which is communicated to the Supervisor.

The *Trajectory Planner* (TP) receives a path from the Supervisor and generates *trajectories* of speeds and yaw angles by considering the vehicle dynamics and physical limits. The trajectory planner is implemented as a model predictive controller that minimizes the deviation between the planned vehicle position along the trajectory and the planned path [17]. Figure 2 shows the assumed relation between new path requests to the TP and how the RCV moves through transition points. The RCV starts at position 0, and once the Supervisor sends a path to the TP the vehicle starts moving. The RCV moves toward a transition point or the goal position and then stops. It is not allowed to move again until a new mission has been received.

The purpose of the *Safe Stop Trajectory Planner* (SSTP) function is to provide a collision free trajectory from an arbitrary initial state in the traffic scene, to a stopped state, preferably outside active lanes. Svensson et al. [14] formulate the safe stop motion planning problem as an optimal

control problem where the safety of the stopping position is considered in the cost function. Also, an algorithm is provided to solve the problem in real time, which is used for the simulations and experiments in this study.

The algorithm is based on selection from a pre-computed set of stopping trajectories. Ahead of run-time, a grid of initial states and candidate terminal states is defined in a Cartesian coordinate system with the RCV at the origin. The grid of candidate terminal states is selected to span the reachable set of stopping positions given the dynamic constraints of the vehicle, and the planning horizon. For each pair of initial and terminal states, an optimal control problem is formulated. If a feasible solution exists for the pair, it is stored in the set of candidate stopping trajectories.

At run-time the library is loaded and the algorithm performs the following steps:

1) A subset of trajectories is selected based on current vehicle velocity.
2) Trajectories in the subset are checked for collisions with respect to an occupancy grid defined in the vehicle frame.
3) A cost based on the location of the final state and time of arrival is evaluated for the collision-free trajectories, and the lowest cost trajectory is selected.

The SSTP receives a request to plan from the Supervisor. The outcome of the planning results in success or failure, which is communicated to the Supervisor.

The *Controller* receives reference trajectories simultaneously from both the Trajectory Planner and the Safe Stop Trajectory Planner. The Supervisor mandates which trajectory is selected for controlling the RCV. It is assumed that the Controller calculates required control commands for the steering angle and acceleration based on the reference trajectory, and that it guarantees a finite bound on deviations from that trajectory. The control commands are passed on to the Vehicle component, where the actuators are controlled.

The Controller may also be commanded by the Supervisor to stop as quickly as possible. This mode will be referred to as *Automatic Emergency Brake* (AEB).

III. Supervisor Design

To accomplish the transport mission, the Supervisor brings together the two path planners SPP and UPP and makes sure that the SSTP is activated by the Controller if and only if necessary, given that the planners and controller guarantee that the assumptions put upon them are fulfilled.

During a nominal parking-to-parking mission (no failure occurs during the mission) the mission is completed as follows. The Supervisor first acquires the current position and the goal position from the Localization component. This information is passed on to the SPP. The SPP responds with the structured path and the two transition points Tp1 and Tp2. The Supervisor requests the UPP to create a path from the current position to Tp1. The path is then passed from the Supervisor to the TP. The Controller is commanded by the Supervisor to control according to the trajectory supplied by the TP. When the RCV moves close to Tp1, the Supervisor concatenates the unstructured path and the structured path. The Supervisor discards the unstructured path and sends only the structured path to the TP after Tp1 has been reached. When the RCV moves close to Tp2, the UPP is requested by the Supervisor to create a path between Tp2 and the goal position. When the final unstructured path has been created, the structured path and the unstructured path are concatenated until Tp2 has been reached. The Supervisor commands the RCV to remain stationary once it arrives and stops at the goal position. This process is repeated once a new mission is given.

If a failure occurs and the nominal operational conditions are disrupted, then the Supervisor activates the SSTP to reach a minimal risk condition. If the SSTP fails, the Supervisor activates the AEB. Activation of the AEB does not in general stop the RCV in a minimal risk condition, but it is considered safer to stop than to continue moving with critical failures.

There are many possible sources of failures in vehicle systems, and this paper considers only a few. The Localization component can detect GPS sensor failures, and communicates the failure state to the Supervisor. Since the planners and the controller require GPS for path tracking, GPS sensor failures are violations of the nominal operational conditions. SPP and UPP may also fail to find paths. If that happens without Supervisor action, then the RCV gets stuck at one of the transition points, which is again not considered nominal operation. Hence, the Supervisor needs to respond to GPS, SPP, and UPP failures and activate the SSTP. Additionally, if the SSTP itself experiences failures, then the AEB shall be activated.

A. Formal requirements

Based on the desired system behavior described above, a number of design requirements are identified and then expressed in natural language.

1) *missionComplete:* The supervisor and the four concurrent control functions shall never stop at invalid end states. The state after a safe or emergency stop is terminal.
2) *stopInTheEnd:* The RCV's speed shall always eventually be zero.
3) *allPathsKnown:* If the vehicle stops safely at the goal position, then all paths must be known. Otherwise the vehicle has been running without a reference, or the model is wrong.
4) *driveOnlyOnPaths:* If the vehicle drives past the endpoint of a path, then the next path must be known. Similarly, the goal position shall never be passed.
5) *safeStop:* If the vehicle stops safely along the road, then an error must have occurred and the SSTP must have executed successfully.
6) *unsafeStop:* If the vehicle stops with emergency braking, then an error must have occurred and the SSTP must have failed.
7) *failure:* If an error occurs, then the RCV must be stopped safely beside the road by SSTP. If SSTP fails, AEB must perform an emergency stop.

For the formal verification to result in either a correctness proof or a counterexample, there has to be a formal specification of the correct behavior. The specifications in this paper are formalized as LTL [24]. The LTL formulas are evaluated over an infinite run π of a finite automaton V. Let π_i represent the *i*-th state along the run and $\pi[i]$ represent the infinite run starting from state π_i. Each state π_i of the run contains a set of atomic properties that are true in that state. Apart from the logic operators used in propositional logic, LTL formulas introduce the temporal operators $\Diamond$ 'eventually' and $\Box$ 'always'. A run π satisfies a formula φ:

- $\pi \vDash \varphi$ iff $\pi[0] \vDash \varphi$

Let θ be an LTL formula, and ψ be an atomic property. The definition of when a run $\pi[i]$, $i \geq 0$, satisfies a formula is given inductively:

- $\pi[i] \vDash \psi$ iff $\psi \in \pi_i$
- $\pi[i] \vDash \neg\varphi$ iff $\pi[i] \nvDash \varphi$
- $\pi[i] \vDash \varphi \vee \theta$ iff $\pi[i] \vDash \varphi$ or $\pi[i] \vDash \theta$
- $\pi[i] \vDash \Box\varphi$ iff $\pi[k] \vDash \varphi$ for all $k \geq i$
- $\pi[i] \vDash \Diamond\varphi$ iff $\pi[k] \vDash \varphi$ for some $k \geq i$

The automaton V satisfies a formula φ if every run satisfies the formula. Any run not satisfying the formula is a *counterexample*. V is a model of the system.

Below, a short comment is given for each design requirement together with its LTL formalization. Propositions in the following LTL formulas are implemented through variables and events to be defined in formal models and code. The semantics of using variables and operators in LTL are found in the documentation of SPIN [8], while specifics of the implementation are found online[1].

missionComplete: One of the desired states Goal, SafeStop, or EmergencyStop as highlighted in Figure 3, shall be reachable at all times. Note that SafeStop and EmergencyStop are terminal states in the RCV software, because repair and software reboot must be performed at these states.

$$\Box\Diamond Goal \vee \Diamond\Box SafeStop \vee \Diamond\Box EmergencyStop \quad (1)$$

stopInTheEnd: This requirement relies on three assumptions: (1) the TP can always stop the vehicle at the end of a path, (2) the SSTP can stop the vehicle if it has no failure, and (3) the AEB can always stop the vehicle.

$$\Box\Diamond speed = 0 \quad (2)$$

allPathsKnown and *driveOnlyOnPaths:* The TP cannot take any responsibility for its input; it has to assume that the input is correct because it has no means of detecting errors. Hence the Supervisor needs to take responsibility for sending correct path information to the TP. The use of *Upp1* and *Upp2* is made to distinguish between the two different planned paths in the source and goal parking lots.

$$\Box(Goal \rightarrow Upp1Success \wedge SppSuccess \wedge Upp2Success) \quad (3)$$

[1] https://github.com/krooken/wasp-des-rcv

$$\Box((pos > 0 \rightarrow Upp1Success) \wedge (pos > Tp1 \rightarrow SppSuccess) \wedge (pos > Tp2 \rightarrow Upp2Success)) \quad (4)$$

safeStop, *unsafeStop*, and *failure:* The safe stop command shall be issued if and only if it is needed. Additionally, the AEB shall only be activated if the SSTP fails.

$$\Box(SafeStop \rightarrow (SensorFailure \vee (\neg SppSuccess \wedge SppRequest) \vee (\neg Upp1Success \wedge Upp1Request) \vee (\neg Upp2Success \wedge Upp2Request) \wedge (SstpSuccess \wedge SstpRequest))) \quad (5)$$

$$\Box(EmergencyStop \rightarrow (SensorFailure \vee (\neg SppSuccess \wedge SppRequest) \vee (\neg Upp1Success \wedge Upp1Request) \vee (\neg Upp2Success \wedge Upp2Request) \wedge (\neg SstpSuccess \wedge SstpRequest))) \quad (6)$$

$$\Box(SensorFailure \rightarrow \Diamond(SafeStop \vee EmergencyStop)) \quad (7)$$

B. Modeling in Stateflow

To solve the transport mission, an FSM which governs the operation of the Supervisor component is proposed in Figure 3. The first step in showing that the FSM fulfills the requirements is using model-based design to implement the Supervisor according to this FSM. Stateflow, which is a toolbox for MATLAB [25], is used for this purpose. A state consists of a name, an entry section, and an exit section. The code in the entry/exit section is executed when the state is entered/exited. When a state is active and the condition on one of its outgoing transitions evaluates to true, then the state may be exited and the next state activated. The semantics are formally described in [25], [26]. In the implementation of the Supervisor, only assignments are used in the states. The Stateflow model is automatically translated to C++ code by MATLAB code generation tools, where variables used as conditions on transitions are regarded as inputs, and variables used in assignments in states are regarded as outputs. A C++ node for ROS provides the generated code with input data, and action is taken on the output to send appropriate actions to the other components in the RCV system.

C. Model checking

A dynamical model of the RCV and its components is created in Simulink, and connected to the Stateflow model of the Supervisor. This setup allows for testing the Stateflow model, subject to the requirements. However, testing can only show presence of errors, not abscence, so regardless of how exhaustively the Stateflow model is tested, only a statistical

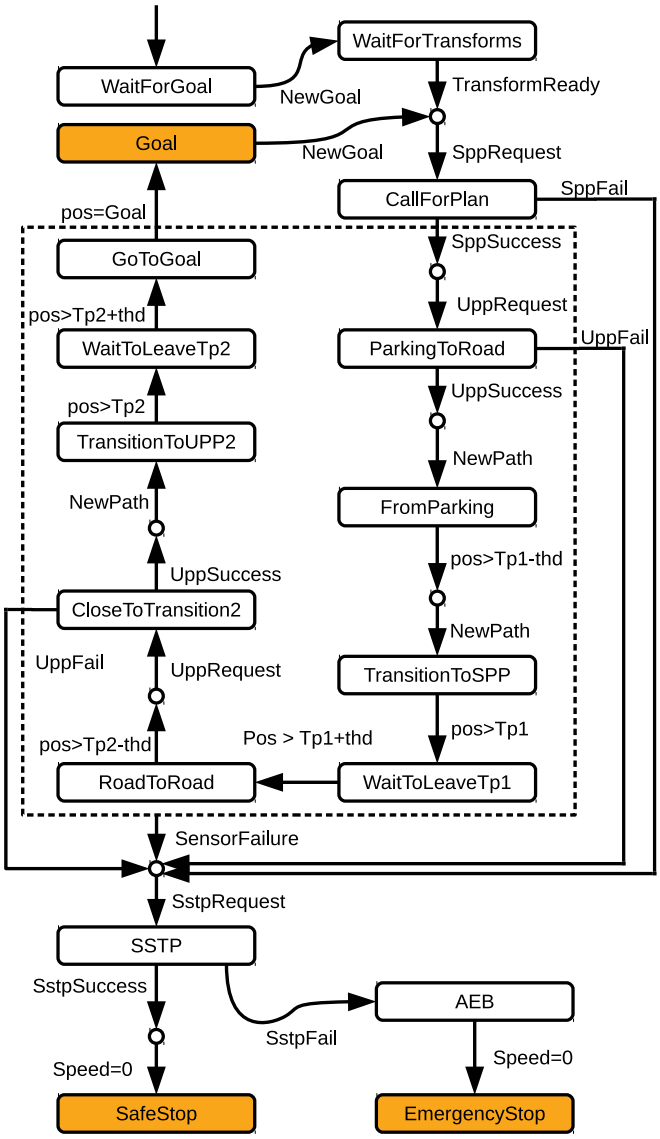


Fig. 3: FSM modeling the Supervisor. Solid rectangles and circles are states. The dashed rectangle is a superstate; all enclosed states have a transition to the SSTP state on sensor failure.

argument can be made for the correctness of the design. To prove absence of errors, the Stateflow model is translated into Promela code and formally verified with the model checker SPIN [8]. Prior research has described methods for translating Stateflow models into Promela code [27], [28], but to our knowledge there is no freely available tool for automatic translation. Since the Stateflow model uses a small and simple subset of the semantics, it is relatively straightforward to do the translation manually.

In addition to a model of the Supervisor, SPIN also needs models of the complete RCV system to prove or disprove specifications. The different components of the RCV are modelled according to the assumptions in Section II and Figure 3. For the verification to be valid for the implementation of the total system, the components other than the Supervisor need to guarantee that their implementations fulfill the assumptions. There are in total six asynchronous software components communicating via asynchronous message queues. The RCV is simulated as a standard discrete-time vehicular longitudinal dynamics model along the length of the path. The Stateflow model and the Promela model are publicly available online.[1] The Promela model has, in addition to the specifications from Section III-A, a specification called *failToReachGoal* that does not allow the system to be in the goal state ($\Box\neg Goal$). SPIN should find a counterexample to this specification, otherwise no transport mission can ever be completed.

IV. Results and Discussion

The results of the case study are presented, interpreted and discussed in two parts. Part one presents the formal verification results and part two presents the experimental results of the system implementation.

A. Formal Verification

SPIN verifies that the Promela model satisfies all specifications except *failToReachGoal*, and the results of the verification can be seen in Table I. The first column refers to the name of the specification, while the third, fourth, and fifth columns show the number of transitions that have been explored during the search, the memory usage during the search, and the time that was required to complete the search, respectively. These numbers give an indication of the proof effort.

The second column specifies what mode SPIN uses to prove the requirement. The safety mode is faster, as can be seen in the table, since SPIN can exploit the proof structure when proving safety specifications, but this technique cannot be used for proving liveness specifications. The first two requirements are the most general, and seen in Table I they are the most demanding ones to prove. In total, it took 247 seconds to prove absence of errors for all requirements. The specification *failToReachGoal* is disproved in 0.6 seconds, verifying that it is indeed possible to reach the Goal state.

TABLE I: Verifying the specifications.

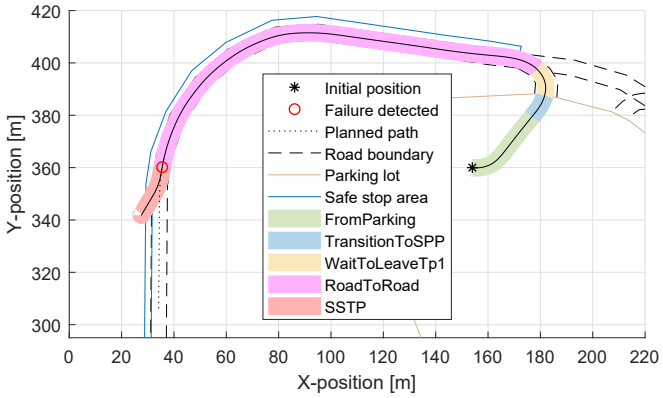
Specification	Type	Transitions	Mem [MB]	Time [s]
missionComplete	Liveness	1.27e+08	680.5	136.0
stopInTheEnd	Liveness	3.50e+07	310.1	38.1
allPathsKnown	Safety	8.02e+06	136.6	10.0
driveOnlyOnPaths	Safety	8.02e+06	136.6	10.0
safeStop	Safety	8.02e+06	136.6	10.0
unsafeStop	Safety	8.02e+06	136.6	10.0
failure	Liveness	2.90e+07	290.0	32.4
failToReachGoal	Safety	5.15e+05	15.0	0.6

B. RCV Experiment

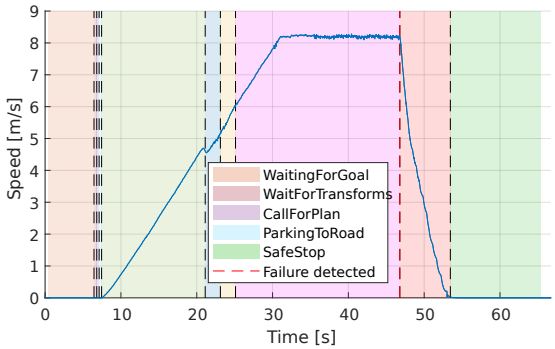
Although SPIN proves that the Promela model satisfies the specifications, several sources might lead to inconsistency between the physical system and the Promela model. To mitigate the risk of discrepancies, a set of scenarios may be run in different verification environments (Promela, Simulink, ROS, and RCV) to see that the same sequence of states are traversed in all of them. A scenario was set up where:

1) The vehicle starts in a parking area, waiting for a mission.
2) The vehicle receives a mission goal, e.g., a pickup request, in the traffic system.
3) The vehicle plans and executes a path through the unstructured area, entering the road network.
4) The vehicle drives along the route toward the goal.
5) A simulated failure is injected into the GPS sensor.
6) The vehicle executes a safe stop maneuver.

Figure 4 illustrates the path (a) and velocity (b) of the vehicle during a Gazebo simulation of the scenario. The colors represent the active state of the FSM. This experiment gives an indication of the usefulness of the proposed method, as the vehicle successfully stopped outside of active lanes promptly after the GPS sensor failure was detected. Also, nominal planner switching between UPP and SPP was handled seamlessly. Figure 5 shows the activation of SSTP during a real world experiment with the RCV[2]. Formally proving absence of deadlocks and reachability of the terminal states reduces the extent to which the supervisor must be verified and is a useful tool in fulfilling the combined requirements for level 4 autonomy from [1] and [4].



(a) A birds eye view of a drive.



(b) Vehicle speed during a drive.

Fig. 4: Results from the simulated scenario in Gazebo. The GPS sensor fails before the RCV has reached the goal, so the SSTP activates and stops the vehicle. Path sections are colored according to current state at that position or time. The coloring reflects the currently active state in Figure 3.

[2]Video source online: `https://vimeo.com/319427372`

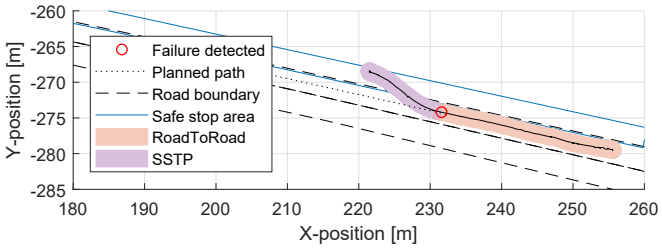


Fig. 5: Results from a real world experiment. SSTP is activated and stops the vehicle in the safest reachable position beside the road.

V. Conclusion and Future Work

This paper studies the problem of selecting an appropriate action in case of hazardous situations during level 4 automated driving. We propose a method based on model-based design, formal methods, and code generation to design a supervisor that arbitrates between planner functions such that a minimal risk condition may always be reached. The method is validated through simulations and experiments. Detection of hazardous internal or external states that should cause safe stop is a non-trivial problem left for future research. Also, the safe stop capability depends on the correctness of the functionality of all other components in Figure 1, which are typically less suitable for verification by formal methods.

The design of the Supervisor, the development of requirements, and the implementation of the Promela model are all manual, which is time consuming and prone to errors. In this regard there is room for improvement. The requirements are difficult to generate automatically, and so is the model of the RCV. However, when the requirements have been developed and the vehicle modeled, the method employed in [9], where correct by construction software is synthesized automatically, could be used to limit the risk of human error.

One weakness of the Supervisor presented in this paper is that, although the correct activation of the Safe Stop Trajectory Planner is guaranteed, the planner may fail to find a suitable trajectory at that particular moment, and resort to the Supervisor to activate the Automatic Emergency Brake, which is likely to stop the vehicle in an active lane. This constitutes an elevated risk level compared to completing a safe stop that terminates by the side of the road. One way to reduce this risk is to store emergency maneuvers, one of which can be activated when the SSTP cannot find new ones [15]. To investigate how the nominal planning behavior could be influenced by the availability of a safe stop maneuver is an item of future work. Other questions for further research include: What other tools are suitable that reduce manual work, while still supporting the features of SPIN? How can it be guaranteed that a safe stop can be performed in the event of an actuator failure?

REFERENCES

[1] SAE, *J3016 Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, Sep 2016.

[2] Waymo. (2017) Waymo safety report - on the road to fully self-driving. [Online]. Available: https://waymo.com/safetyreport/

[3] General Motors. (2018) General Motors self-driving safety report. [Online]. Available: http://www.gm.com/mol/selfdriving.html

[4] ISO/TC 22/SC 32, "ISO 26262: Road vehicles – functional safety," International Organization for Standardization, Tech. Rep., 2012.

[5] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.

[6] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer Publishing Company, Incorporated, 2010.

[7] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[8] G. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, 1st ed. Addison-Wesley Professional, 2003.

[9] T. Korssen, V. Dolk, J. van de Mortel-Fronczak, M. Reniers, and M. Heemels, "Systematic model-based design and implementation of supervisors for advanced driver assistance systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, no. 99, pp. 1–12, 2017.

[10] S. Sims, R. Cleaveland, K. Butts, and S. Ranville, "Automated validation of software models," in *16th Annual International Conference on Automated Software Engineering (ASE)*, 2001, pp. 91–96.

[11] A. Zita, S. Mohajerani, and M. Fabian, "Application of formal verification to the lane change module of an autonomous vehicle," in *13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug 2017, pp. 932–937.

[12] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive cruise control: Hybrid, distributed, and now formally verified," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6664 LNCS. Springer, Berlin, Heidelberg, 2011, pp. 42–56. [Online]. Available: http://link.springer.com/10.1007/978-3-642-21437-0_6

[13] J. Salvado, L. M. M. Custódio, and D. Hess, "Contingency planning for automated vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 2853–2858.

[14] L. Svensson, L. Masson, N. Mohan, E. Ward, A. P. Brenden, L. Feng, and M. Törngren, "Safe stop trajectory planning for highly automated vehicles," *29th IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[15] S. Magdici and M. Althoff, "Fail-safe motion planning of autonomous vehicles," in *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2016, pp. 452–458.

[16] O. Wallmark, M. Nybacka, D. Malmquist, M. Burman, P. Wennhage, and P. Georen, "Design and implementation of an experimental research and concept demonstration vehicle," in *IEEE Vehicle Power and Propulsion Conference (VPPC)*, Oct 2014, pp. 1–6.

[17] S. Kokogias, L. Svensson, G. C. Pereira, R. Oliveira, X. Zhang, X. Song, and J. Mårtensson, "Development of platform-independent system for cooperative automated driving evaluated in GCDC 2016," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, no. 99, pp. 1–13, 2017.

[18] G. C. Pereira, L. Svensson, P. F. Lima, and J. Mårtensson, "Lateral model predictive control for over-actuated autonomous vehicle," in *IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 310–316.

[19] (2017) Open Source Robotics Foundation cardemo. [Online]. Available: https://github.com/osrf/cardemo

[20] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[21] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *IEEE Intelligent Vehicles Symposium Proceedings (IV)*, June 2014, pp. 420–425.

[22] K. Kutzer, "Path planning in unstructured environments: A real-time Hybrid A* implementation for fast and deterministic path generation for the KTH Research Concept Vehicle," Master's thesis, KTH, Royal Institute of Technology, 2016.

[23] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," Stanford University, Tech. Rep., 01 2008.

[24] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning About Systems*. New York, NY, USA: Cambridge University Press, 2004.

[25] MathWorks, https://www.mathworks.com/products/matlab.html, 2017.

[26] A. Tiwari, "Formal semantics and analysis methods for Simulink Stateflow models," SRI International, Tech. Rep., 2002. [Online]. Available: http://www.csl.sri.com/~tiwari/stateflow.html

[27] L. Feng, D. Chen, H. Lönn, and M. Törngren, "Verifying system behaviors in EAST-ADL2 with the SPIN model checker," in *IEEE International Conference on Mechatronics and Automation (ICMA)*, Aug 2010, pp. 144–149.

[28] C. Yamada and D. M. Miller, "Using SPIN to check Simulink Stateflow models," in *IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, June 2015, pp. 161–166.